frontiers in Neuroscience

ORIGINAL RESEARCH
published: 26 August 2021
doi: 10.3389/fnins.2021.713054

Digital Implementation of Oscillatory Neural Network for Image Recognition Applications

*Madeleine Abernot[1], Thierry Gil[1], Manuel Jiménez[2], Juan Núñez[2], María J. Avellido[2], Bernabé Linares-Barranco[2], Théophile Gonos[3], Tanguy Hardelin[3] and Aida Todri-Sanial[1]**

[1] Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, University of Montpellier, CNRS, Montpellier, France, [2] Instituto de Microelectronica de Sevilla, IMSE-CNM, CSIC, Universidad de Sevilla, Sevilla, Spain, [3] A.I. Mergence, Paris, France

Computing paradigm based on von Neuman architectures cannot keep up with the ever-increasing data growth (also called "data deluge gap"). This has resulted in investigating novel computing paradigms and design approaches at all levels from materials to system-level implementations and applications. An alternative computing approach based on artificial neural networks uses oscillators to compute or Oscillatory Neural Networks (ONNs). ONNs can perform computations efficiently and can be used to build a more extensive neuromorphic system. Here, we address a fundamental problem: can we efficiently perform artificial intelligence applications with ONNs? We present a digital ONN implementation to show a proof-of-concept of the ONN approach of "computing-in-phase" for pattern recognition applications. To the best of our knowledge, this is the first attempt to implement an FPGA-based fully-digital ONN. We report ONN accuracy, training, inference, memory capacity, operating frequency, hardware resources based on simulations and implementations of 5 × 3 and 10 × 6 ONNs. We present the digital ONN implementation on FPGA for pattern recognition applications such as performing digits recognition from a camera stream. We discuss practical challenges and future directions in implementing digital ONN.

Keywords: artificial intelligence, auto-associative memory, FPGA implementations, learning rules, oscillatory neural networks, pattern recognition

OPEN ACCESS

Edited by:
Chunsheng Jiang, China Academy of Engineering Physics, China

Reviewed by:
Lyes Khacef, University of Groningen, Netherlands
Timothée Levi, Université de Bordeaux, France

****Correspondence:***
Aida Todri-Sanial
aida.todri@lirmm.fr

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *21 May 2021*
Accepted: *04 August 2021*
Published: *26 August 2021*

Citation:
Abernot M, Gil T, Jiménez M, Núñez J, Avellido MJ, Linares-Barranco B, Gonos T, Hardelin T and Todri-Sanial A (2021) Digital Implementation of Oscillatory Neural Network for Image Recognition Applications. Front. Neurosci. 15:713054. doi: 10.3389/fnins.2021.713054

1. INTRODUCTION

In recent years, we have witnessed a proliferation of smart edge devices adopted by all industry sectors such as smart homes, smart city cameras, smart autonomous driving cars, smart healthcare, smart manufacturing, etc. Most edge devices are getting smaller and compact.

Using Artificial Neural Networks (ANNs), specifically Deep Neural Networks (DNNs) to create Artificial Intelligence (AI) at the edge, has successfully been used to teach smart systems to recognize or detect objects (Redmon et al., 2016; Krizhevsky et al., 2017; Shah and Kapdi, 2017; Yang and Song, 2018; Jiao et al., 2019), read texts (Jackel et al., 1991), and understand speeches (Xiong et al., 2018; Nassif et al., 2019). Contraints to such applications on edge devices derives from the inherent limitations of power consumption, memory, and little to no bandwidth. In addition, privacy and security concerns would recommend the data to be stored locally. In contrast, ANNs or DNNs systems that enable AI at the edge are getting larger to cope with the ever-increasing amount of data. Thus, resulting in more power consumption, memory, and bandwidth demand.

Systems based on ANNs and Convolutional Neural Networks (CNNs) running on traditional von Neumann architectures require a large amount of memory, computational power, and bandwidth demand. While they perform well on expensive hardware such as GPUs (Pham et al., 2019), they are often unsuitable for smaller edge devices. Such a disconnect between the growing need in AI at the edge and limitations of processing hardware has compelled significant research efforts into beyond-von Neumann systems such as neuromorphic computing paradigms deployable at the edge (Bey, 2020; Kendall and Kumar, 2020).

This paper focuses on an alternative, low-power, neuromorphic computing approach with Oscillatory Neural Networks (ONNs) (Raychowdhury et al., 2019; Csaba and Porod, 2020). The ONN is a system of coupled oscillators mimicking at circuit level the basic structure of the brain architecture. The key feature of ONNs is to encode the information on the phase relations between oscillators and to let them oscillate using their physical dynamics to compute. For example, the random start of five metronomes (similar to grandfather's clock) will make them oscillate in parallel (Met, 2013). After several cycles, they get synchronized in frequency while their phase relations can tell us if they are in- or out-of-phase. Contrarily to the classical computation based on voltage amplitude to determine a logic "1," or "0," in ONN we use the phase relations to determine the logic "1" (out-of-phase 180^o) or "0" (in-phase 0^o). Thus, working with parallel oscillators in the frequency and phase domains allows to reach fast and low-power computation (Roychowdhury, 2014; Shukla et al., 2016). This makes ONN an ideal solution to bring artificial intelligence on edge devices.

ONN principle was first introduced in Hoppensteadt and Izhikevich (2000) where ONN showed good associative memory properties. Thus, there is a recent interest to exploit ONN for large-scale associative memory applications. While there is a lot of ongoing research on devices and analog architectures to implement ONN efficiently (Csaba and Porod, 2013; Jackson et al., 2015; Shi et al., 2016; Kumar and Mohanty, 2017; Corti et al., 2019; Velichko et al., 2019), we focus on addressing a more fundamental problem—can we perform relevant AI applications such as image recognition with ONN? We explore ONN at small-scale (up to 60 coupled oscillators) as a first attempt to show phase computation in the digital domain. Despite being a small-scale ONN, we investigate advantages and limitations on image recognition tasks suitable for AI applications on edge devices. To do so, we implement an FPGA-based digital ONN to serve as a proof-of-concept of the ONN computing paradigm for enabling AI at the edge.

The rest of the paper is organized as follows. In section 2, we present materials and methods used for all experiments carried out for this work. In section 2.1, we introduce the ONN model and compare it with state-of-the-art associative memory models. Then, in section 2.2, we present the training methods we apply to the ONN. Afterward, in section 2.3, we describe the digital ONN design. section 2.4 presents methods used for ONN validation and characterization for design simulation and FPGA implementation. Next, section 2.5 exposes methods for a 10×6 ONN used for image recognition from a camera stream. section 3 reports on results related to ONN simulation, ONN FPGA implementation, and image recognition application using such 10×6 ONN. Finally, in section 4, we discuss the advantages and limits of ONN and future directions.

2. MATERIALS AND METHODS

2.1. ONN Biological Inspiration and Related Works

Recent researches on brain-inspired computing paradigms have focused on Spiking Neural Networks (SNNs) (Maass, 1997; Paugam-Moisy and Bohte, 2012; Zenke and Ganguli, 2018). Neurons in the brain use voltage spikes to transmit information and SNNs emulate spikes to compute efficiently in the time domain. On another side, the human brain's electrical activity has shown macroscopic oscillations observable on electroencephalogram signals (EEG). Since then, on going research has tried to find relationships between oscillation behavior and neuronal activity (Martindale, 1978; Lieff, 2012). ONNs are a novel computing paradigm that uses coupled oscillators as neurons to mimic brain wave oscillations. With ONNs, we exploit the synchronization dynamics of physical oscillators to compute (Hoppensteadt and Izhikevich, 1997).

In ONNs, information is encoded in the phase of the oscillators. Selecting one oscillator as the reference, we can use each oscillator to encode frame values into phases (between 0° and 180°). Computation consists of oscillators initialization with initial phase state ϕ_{init} ($0^o < \phi_{init} < 180^o$). Then, oscillators will oscillate in parallel within multiple states until they stabilize and lock in phase. Once they stabilize, we can measure phase information of the final state ϕ_{end} and associate it to output frame values. When synchronization occurs, the oscillators oscillates in parallel so it permits fast computation, independently from the number of oscillators. Additionally, computing with oscillators in the frequency domain allows low voltage operation. These two features allow for low power computation (Roychowdhury, 2014; Shukla et al., 2016), and are attractive to implement artificial intelligence on edge devices.

Thus, ONN advantages come from the analog-based computing principle and the hardware implementation. Different approaches have been explored to emulate oscillating neurons in ONN architectures such as using spin-torque oscillators (Csaba and Porod, 2013), ring oscillators (Csaba et al., 2016), microelectromechanical oscillators (Kumar and Mohanty, 2017), PLL-based oscillators (Shi et al., 2016), or more recently, beyond-CMOS devices such as VO_2-based oscillators (Corti et al., 2019). In parallel, different analog couplings have been proposed using resistors and capacitors (Csaba and Porod, 2020). In literature, (Ahmed et al., 2021) reported a fully coupled ONN implementation of 100 neurons. It is, to the best of our knowledge, the largest implementation of an ONN with fully coupled oscillators. Other efforts have been deployed on the hardware implementation of ANNs (Misra and Saha, 2010; Levi et al., 2018) and SNNs have shown a particular interest for hardware implementations due to their low-power properties. Some authors have developed FPGA-based platforms

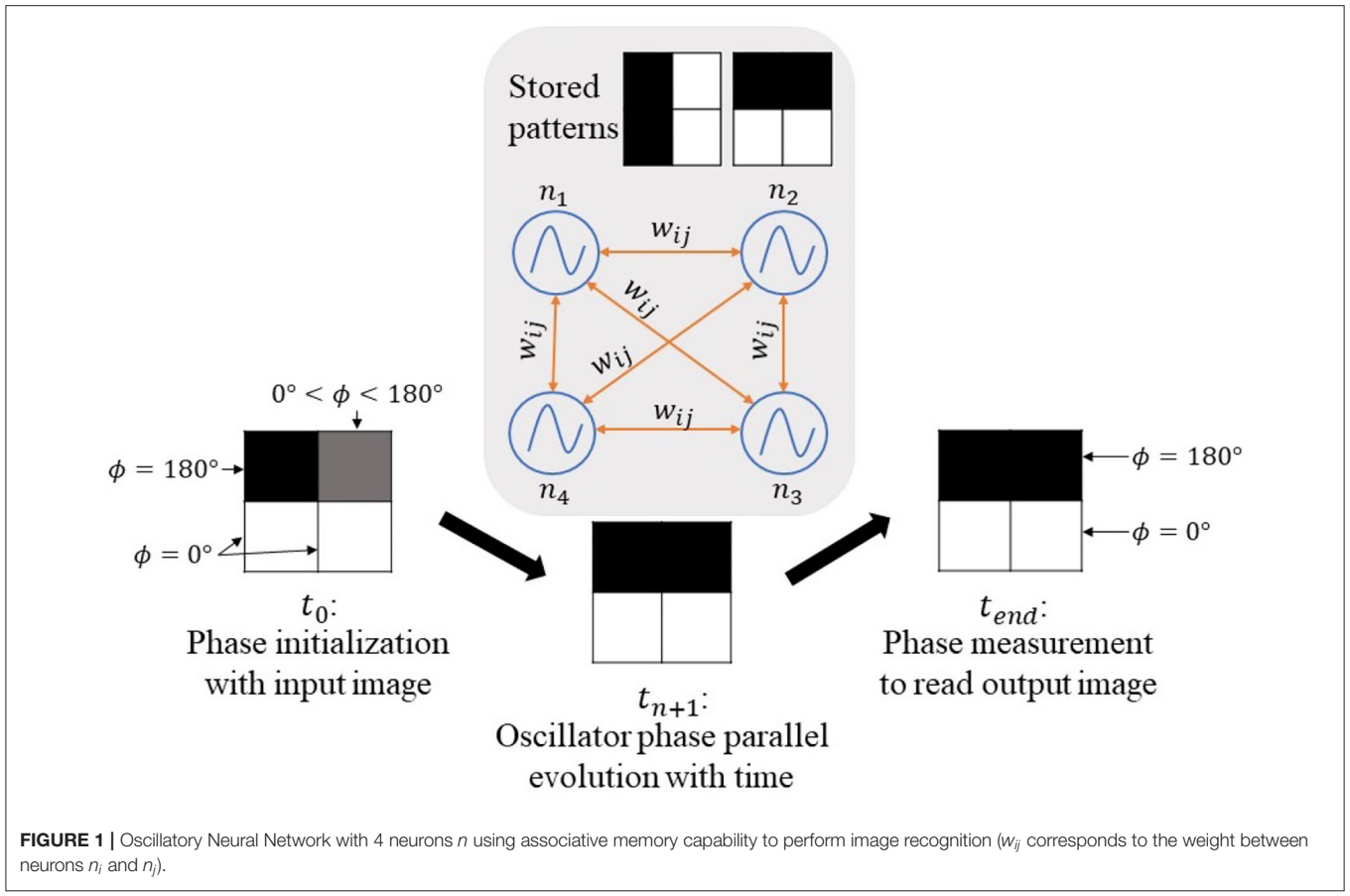

FIGURE 1 | Oscillatory Neural Network with 4 neurons *n* using associative memory capability to perform image recognition (w_{ij} corresponds to the weight between neurons n_i and n_j).

with SNNs (Rosado-Muñoz et al., 2012; Guo et al., 2021), and mostly perform image classification tasks (Han et al., 2020; Xia et al., 2020). Some others have focused more on customizable neuromorphic chips (Khan et al., 2008; Mitra S, 2009; Ma et al., 2017; Davies et al., 2018). In this manuscript, we present a first proof-of-concept of the ONN paradigm implemented on FPGA, and further efforts are needed to develop ONN-based hardware implementations at the same scale as existing SNN neuromorphic chips.

ONNs have shown associative memory computing properties (Hoppensteadt and Izhikevich, 2000), like the one possessed by Hopfield Neural Networks (HNNs) (Hopfield, 1982). Associative memory systems can store patterns and associate each possible input to one of the stored patterns. Stored patterns represent the minima of an energy function toward which the network evolves. In the case of multiple stored patterns, the network will evolve and converge to the nearest energy minimum, meaning the closest stored pattern from the input. We train the ONN to store patterns by adapting the coupling between oscillators. The coupling elements represent the weights of the oscillators. If we consider the image processing domain, each pixel is an oscillator, and the phase information represents the pixel color. We compute weights corresponding to training images stored in ONN. Thus, when we initialize the network with a new image, it converges to the closest stored image, see **Figure 1**. This is what we define by image recognition task in this paper. In some cases, the network evolves between states without stabilizing, meaning the network does not converge.

HNNs have shown capabilities to work with binary/bipolar-valued patterns (Hopfield, 1982), continuous-valued patterns (Ramsauer et al., 2021), and complex-valued patterns (Muezzinoglu et al., 2003; Tanaka and Aihara, 2009). In this work, we use binary patterns representing in- and out-of-phase relations and corresponding to black and white images.

2.2. ONN Learning

The Hebbian learning rule (Morris, 1999) is one of the most popular learning algorithm to calculate synaptic weights for bipolar-valued stored patterns on HNNs. However, the Storkey learning rule (Storkey et al., 1997) has been reported to possess higher storage capacity and robustness against correlated stored patterns and crosstalk phenomenon (Storkey, 1997; Wu et al., 2012). So, we apply both the Hebbian and the Storkey rules to the ONN and compare results.

For both learning rules, we transform each stored pattern with index k into a vector ξ^k of length N, with N, the number of neurons inside the network. Each vector element is bipolar (−1/1). For the Hebbian rule, synaptic weight w_{ij} between neuron n_i and neuron n_j is calculated as:

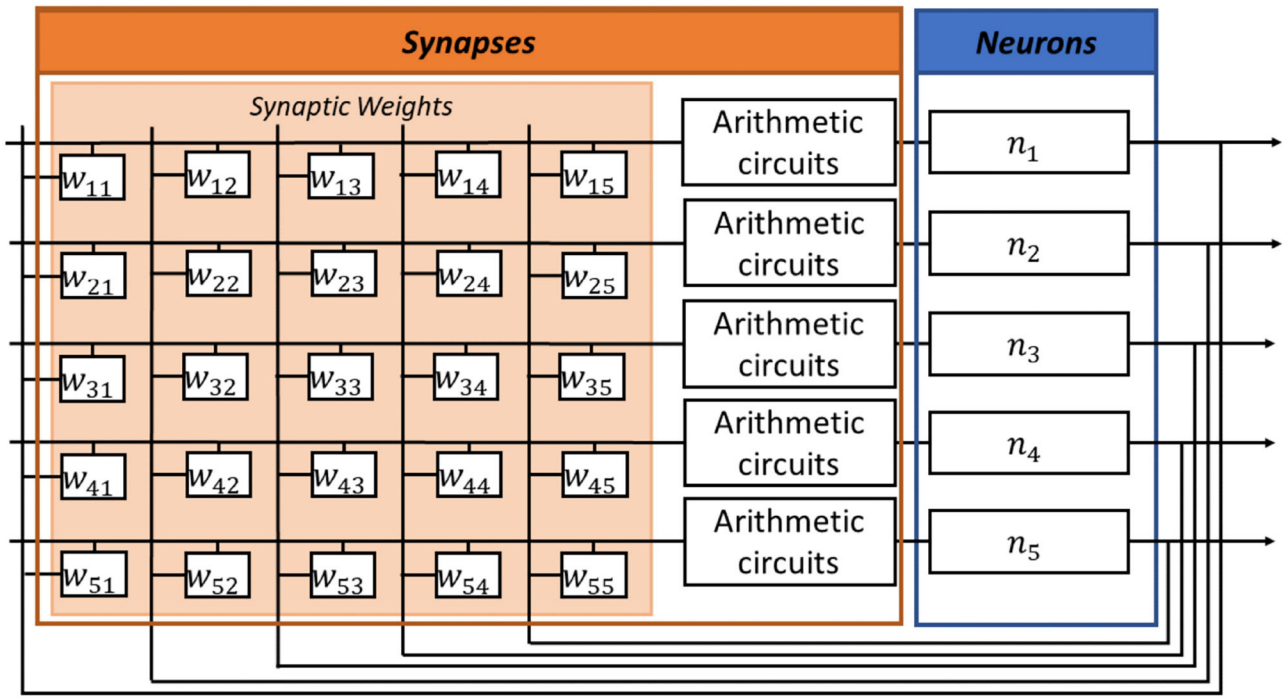

FIGURE 2 | Digital ONN architecture with fully parallel implementation of neurons and a combinatorial implementation of the arithmetic synapse block.

$$w_{ij} = \sum_k \xi_i^k \xi_j^{k^T} \tag{1}$$

with $w_{ij} = 0 \; \forall \; i = j$.

Whereas, Storkey learning rule is defined as:

$$w_{ij}^{\nu} = w_{ij}^{\nu-1} + \frac{1}{N}\xi_i^{\nu}\xi_j^{\nu} - \frac{1}{N}\xi_i^{\nu}h_{ji}^{\nu} - \frac{1}{N}h_{ij}^{\nu}\xi_j^{\nu} \tag{2}$$

$$h_{ij}^{\mu} = \sum_{k=1,k\neq i,j}^{N} w_{ik}^{\mu-1}\xi_k^{\mu} \tag{3}$$

where $w_{ij}^0 = 0 \; \forall \; i, j$, and h_{ij} is a form of local field at neuron i.

Theoretically, the Hebbian memory capacity, represented by the maximum number of stored patterns K is derived in Amit et al. (1987) as

$$K = 0,14 * N \tag{4}$$

For our simulations and implementations, we apply up to K patterns. We calculate weights off-line using a software algorithm and we store them in our digital design.

2.3. Digital ONN Design

We develop a digital ONN as a proof of concept of the computing in phase paradigm to explore ONN architectures and AI at-the-edge applications. We present an ONN digital design inspired by hybrid analog-digital work from Jackson et al. (2019) but without analog components.

In Jackson et al. (2019), synapses are implemented by a resistor network, and a critical analog comparator is required at the input of each digital neuron. In contrast, in our digital ONN implementation, we use an arithmetic circuit for each synapse and there is no analog comparator in neurons. **Figure 2** illustrates the digital ONN architecture. It is a fully parallel design, meaning we implement each neuron inside the FPGA to oscillate in parallel. In addition, the synapses block is combinatorial, which means all synaptic operations are computed in parallel. Also, our architecture needs extra blocks to control synapses and neuron signals. **Figure 3** presents our digital ONN design composed of neurons, synapses, and control blocks. In the following subsections, we detail the implementation of each block.

2.3.1. Neuron Block

In this design, ONN neurons are phase-changed oscillators. Each neuron n computes the phase difference between the present input and output oscillations to align the output in-phase with the input. All neurons are identical according to the diagram in **Figure 3**. A neuron has one 1-bit input, n_{in}, and one 1-bit output, n_{out}, oscillating signals (square signals), in addition to synchronization, initialization, and control signals. The synapse block generates the n_{in} signal, which determines the evolution of the neuron phase. The ϕ_{out} and *state_changed* signals give information on the neuron phase and its evolution. We use *full_tick*, *ser_state_in*, and *ser_state_out* signals to initialize the neuron phase, and *reset*, *clk*, and *slowclock* signals to ensure synchronization.

The process starts with the initialization of the output signal phase ϕ_{out}. It triggers the initial output oscillation n_{out} corresponding to ϕ_{out}. Automatically, the synapses block

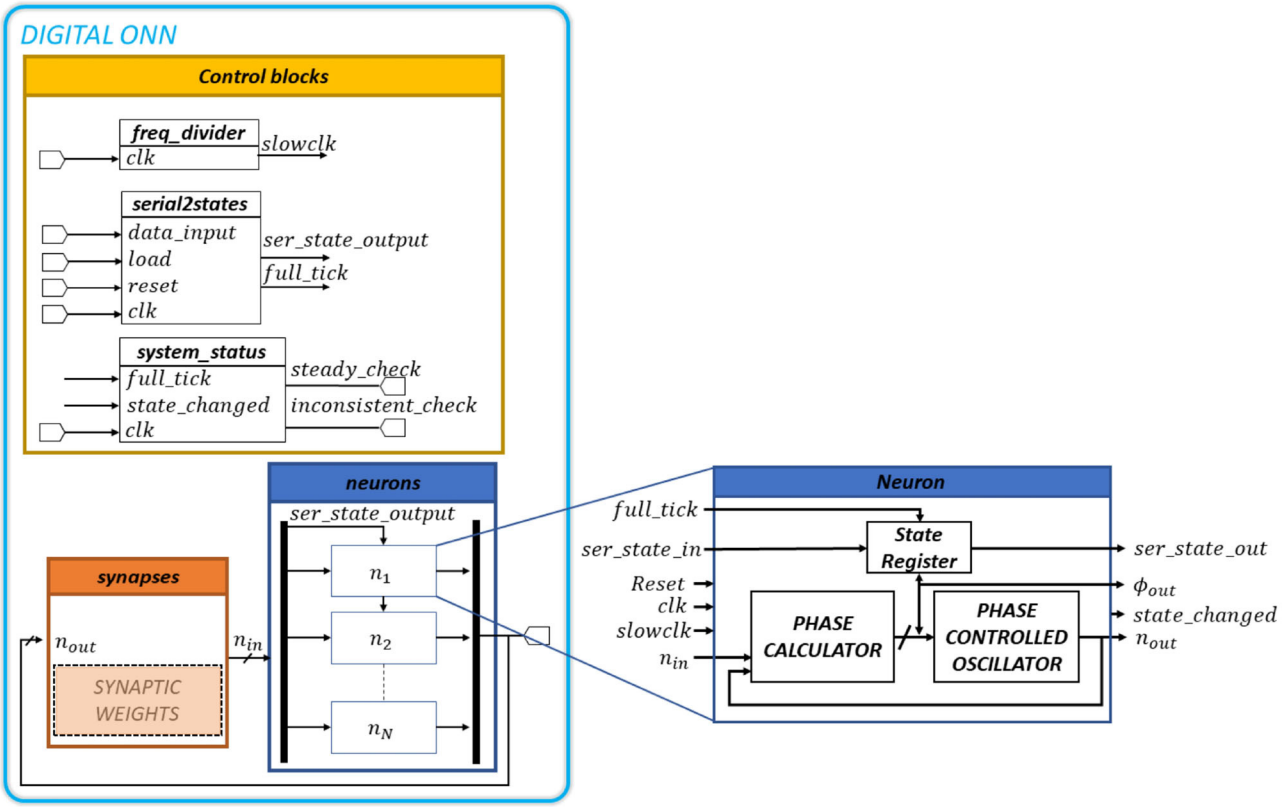

FIGURE 3 | Complete digital ONN design representing digital blocks and block description with the operating principle of a single oscillatory neuron.

computes the new input oscillation n_{in} with the phase ϕ_{in}. We initialize all neurons serially when *full_tick* is activated by connecting *ser_state_out* signal to *ser_state_in* signal of the neighbor neuron. Note, when initialization is over, a scan-path between *ser_state_out* and *ser_state_in* is configured to load and read ONN's state in series.

Then, each neuron calculates the phase difference $\Delta\phi$ between ϕ_{in} and ϕ_{out} and uses it to update the new ϕ_{out} with a phase calculator block. The phase calculator block contains two edge detectors and a finite state machine (FSM). Edge detectors detect rising edges on n_{in} and n_{out} oscillating signals. FSM measures the time difference between n_{in}'s rising edge and n_{out}'s rising edge to define $\Delta\phi$. The $\Delta\phi$ value allows us to update the neuron output phase ϕ_{out} aligning n_{out} signal with n_{in} signal, as:

$$\phi_{out} = \phi_{out} + / - \Delta\phi \tag{5}$$

Note, the sign $(+/-)$ depends on the first rising edge detected. $(-)$ if n_{out}'s rising edge is detected first and $(+)$ if n_{in}'s rising edge is detected first. Note, the n_{in} signal phase is set by the weighted sum of the neuron's input signals.

Finally, we apply the new ϕ_{out} to the oscillating output signal n_{out} with a phase-controlled oscillator. The phase-controlled oscillator contains a circular shift register with a multiplexer. The shift register has 16 stages to represent square signals with different phases so, 16 phase options are available. The 16-bit pattern [1111111100000000] cycles continuously through time. So, through the multiplexer selection bits, we select a shift-register state corresponding to a square signal with a distinct phase. This square signal becomes the new neuron output n_{out}. **Figure 4A** shows the logic diagram of the phase-controlled oscillator and **Figure 4B** the waveforms corresponding to stage 0 (in-phase, $\phi_{out} = 0^{\circ}$) and stage 2 (out-of-phase, $\phi_{out} = 45^{\circ}$). The register controlling the multiplexer stores the neuron state, or equivalently the phase of the neuron output. Note, we use different clocks (driven by the system clock) to control the state register and the shift register. The latter is driven by a slow clock generated from the system clock so that the multiplexer's output cycles as long as its control register remains unchanged with a period $T_{osc} = 16 * T_{slowclk}$.

2.3.2. Synapses Block

ONN synapses block contains weights and computes each neuron input oscillation n_{in}. We use an arithmetic logic circuit for our digital ONN design to generate the input signal to the i-th neuron as:

$$n_{in}[i] = sign(\sum_{j} w_{ij} - \sum_{k} w_{ik}) \tag{6}$$

where j extends to those neurons with $n_{out}[j] = 1$ and k to those with $n_{out}[k] = 0$. It is the most expensive component in terms

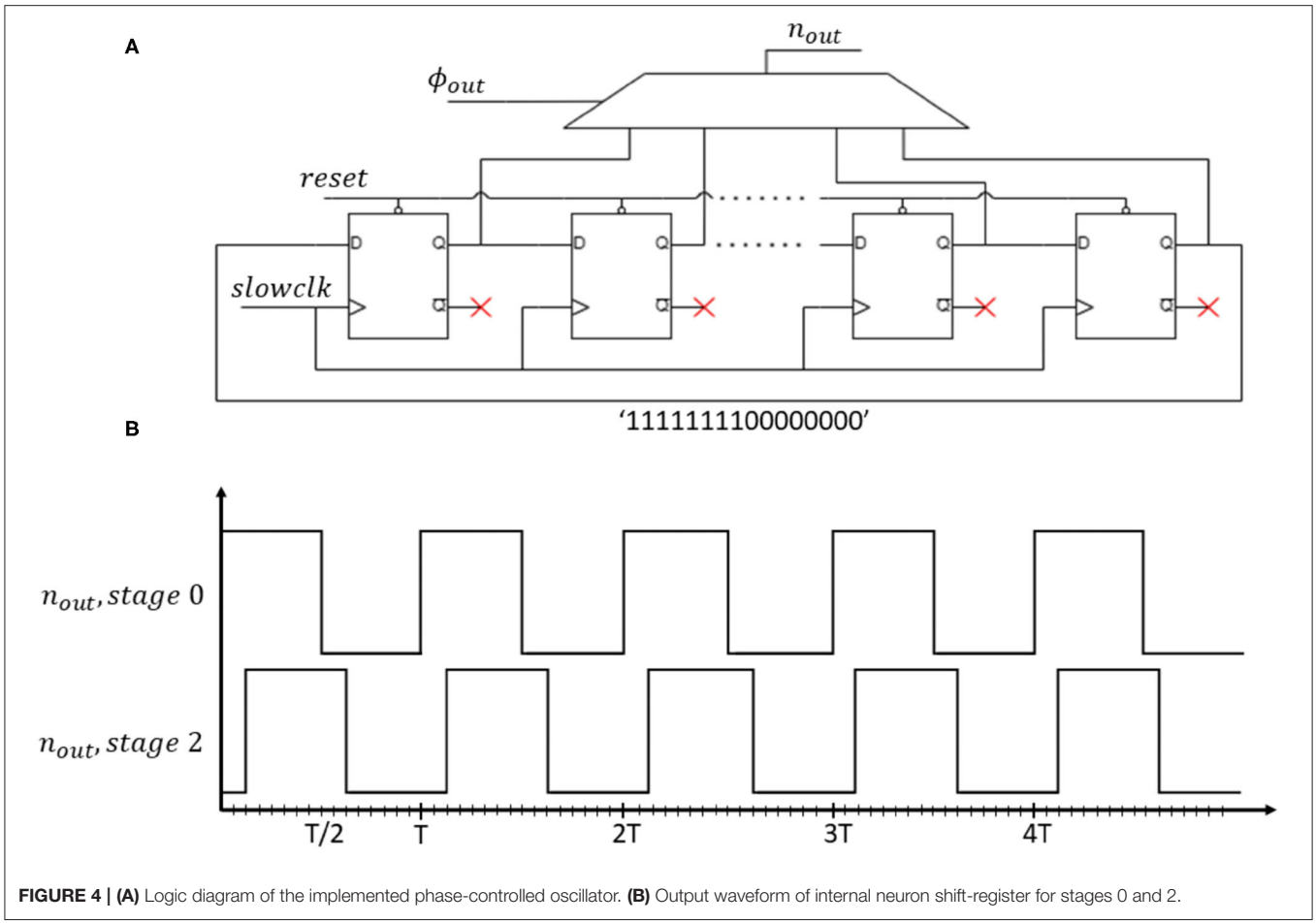

FIGURE 4 | (A) Logic diagram of the implemented phase-controlled oscillator. **(B)** Output waveform of internal neuron shift-register for stages 0 and 2.

of resources. Results in this paper correspond to a combinatorial synapses block using 5-bit weights.

2.3.3. Control Block

In addition to neurons and synapses, our digital design requires a control block to control and monitor ONN computation. It is mainly in charge of three tasks. (1) The initialization step required to carry out an ONN computation. We serially apply input state with a scan-path on neuron's state registers while activating the *full_tick* signal. (2) The control block generates a slow clock to ensure ONN operation. The relation between the slow clock and the system clock is $T_{slowclk} = 4 * T_{clk}$. We use a frequency divider of 2-bit length to speed up the system performance. (3) The generation of the steady (*steady_check*) and the inconsistent (*inconsistent_check*) signals (see **Figure 3**). They indicate wether ONN gives a stable or unstable state. We activate the steady signal once the ONN reaches a stable state, meaning all neuron phases ϕ_{out} are stable for two oscillation periods (T_{osc}). We activate as well the inconsistent signal if the ONN does not achieve any stable state after a time (160 μs) arbitrarily defined by us. To do so, the control block monitors neurons' oscillation activity.

The combination of neuron blocks, synapses block, and control block creates our complete fully-digital ONN design. Next, we carry out tests to validate the associative memory properties of our design, and its characteristics.

2.4. ONN Characterization Methods

To validate our digital ONN design, we characterize 5 × 3 and 10 × 6 digital ONNs using both Hebbian and Storkey learning rules. First, in section 2.4.1, we present simulation methods used to evaluate its performances using software tools. Next, in section 2.4.2, we present FPGA implementation methods.

2.4.1. Simulation

We validate and characterized ONN with simulation software tools using a testbench before being implemented on FPGA. We use a testbench on the software Xilinx's Vivado Design Suite 2018.2. We also perform post-place&route simulations to characterize ONN designs following Vivado's default strategies for synthesis and implementation. We set the target device to the Xilinx 7-series FPGA, the XC7Z020-1CLG400C since it is used for implementation afterward.

We carry out simulations with a 5 × 3 ONN and a 10 × 6 ONN configured for pattern recognition using both Hebbian and Storkey learning rules. We configure the 5 × 3 ONN design with three stored patterns with standard 5 × 3 bitmap representations of digits 0, 1, and 2 (see **Figure 5A**). Each

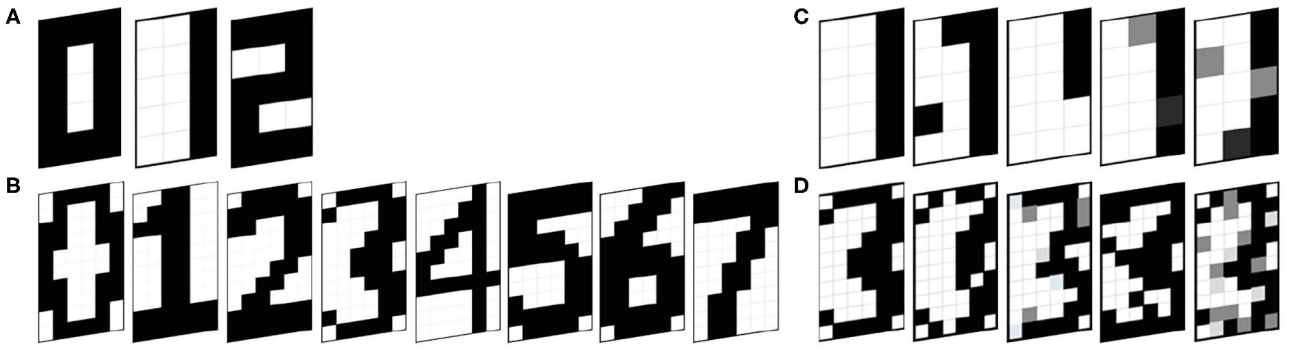

FIGURE 5 | Reference patterns for **(A)** 5 × 3 ONN, **(B)** 10 × 6 ONN, and Test sets **(C)** associated to digit 1 for the 5 × 3 ONN, **(D)** associated to digit 3 for the 10 × 6 ONN.

pixel of the image corresponds to a neuron. Each pixel color is associated with each neuron phase, with white as in-phase (0°), and black as out-of-phase (180°). Gray-level pixels are encoded with intermediate phases. Similarly, we configure the 10 × 6 ONN design with five stored patterns representing digits 0, 1, 2, 3, and 4 (see **Figure 5B**). We use two test sets, one for each ONN, containing both stored patterns and corrupted patterns. We create four corrupted patterns associated with each stored pattern by changing several pixel values with opposite or intermediate values (black or white or gray), see **Figures 5C,D**. We use Hamming Distance (HD) as a metric to measure the corrupted patterns' deviation from the stored ones. The HD between two patterns ξ^{ν} and ξ^{μ} of i elements is defined as:

$$HD = \frac{1}{2}\sum_{i}(\xi_i^{\nu} - \xi_i^{\mu}) \tag{7}$$

In our test set, each stored pattern has four associated corrupted ones that are closer in their Hamming distances than any other stored patterns. Corrupted patterns are supposed to stabilize on the stored pattern with closer HD.

2.4.2. FPGA Implementation

Once we validated and characterized the ONN design using simulation, we implement it on an FPGA to ensure ONN operation on a real embedded platform and to measure real performances. Here, we describe the experimental set-up necessary for ONN implementation on FPGA.

We test real pattern recognition performances of 5 × 3 and 10 × 6 digital ONN designs by implementing them on an FPGA chip. We choose to use a Zybo-Z7 Digilent development board (Digilent, 2018). The board has many communication ports, memory spaces, user interaction tools, and a Xilinx Zynq-7000 System on Chip (SoC). The SoC integrates a dual-core ARM Cortex-A9 processor with Xilinx 7-series FPGA, the XC7Z020-1CLG400C. Only FPGA resources are necessary for the digital ONN implementation. As for simulation, we use Xilinx's Vivado Design Suite 2018.2 software to implement the digital ONN design on FPGA.

Figure 6 shows the system level architecture, including the digital ONN design, for performing pattern recognition on FPGA. The architecture includes the digital ONN described in section 2.3 and a scheduler block to control it. The scheduler has four control blocks to monitor and check the ONN operations. First, the system clock is divided inside the slow clock block to ensure the operations. Test patterns are stored inside the ONN controller and we use switches to select the input pattern. Next, the controller sends the input pattern to the ONN and waits until the end of ONN computation (steady signal activated). In the end, the ONN controller measures the ONN output state, applies a mask to identify the stored image, and the LED controller block turns *on*/*off* the corresponding LEDs, indicating which stored image ONN retrieves. The development board provides switches and LEDs needed by the architecture.

First, we validate our ONN FPGA implementation by performing the same tests as in simulation, and we compare results. We define a training configuration with two parameters; the learning rule and the stored pattern combination. Respecting the Hebbian maximum capacity (stored pattern limit) described in section 2.2, we try multiple stored pattern combinations for both Hebbian and Storkey learning rules for the 5 × 3 ONN and the 10 × 6 ONN, see **Table 1**. Similar to simulation characterization, the test set includes stored patterns plus four corrupted versions of each. See examples on **Figure 5**.

We also test multiple frequencies to compare with the maximum frequency evaluated in simulation. ONN input frequency is defined by the system clock of the development board (125 MHz) divided by a configurable factor. The configurable factor allows us to divide the frequency by multiples of 2. First, we set the ONN input frequency to 7.8125 MHz which is much lower than the frequency estimated by simulation static timing analysis. Then, we modify the configurable factor to try higher frequencies, up to the maximum frequency 125 MHz.

We use the error rate (E_R) metric to check the ONN operation. It is computed as:

$$E_R = \frac{\epsilon}{I_{tests}} \tag{8}$$

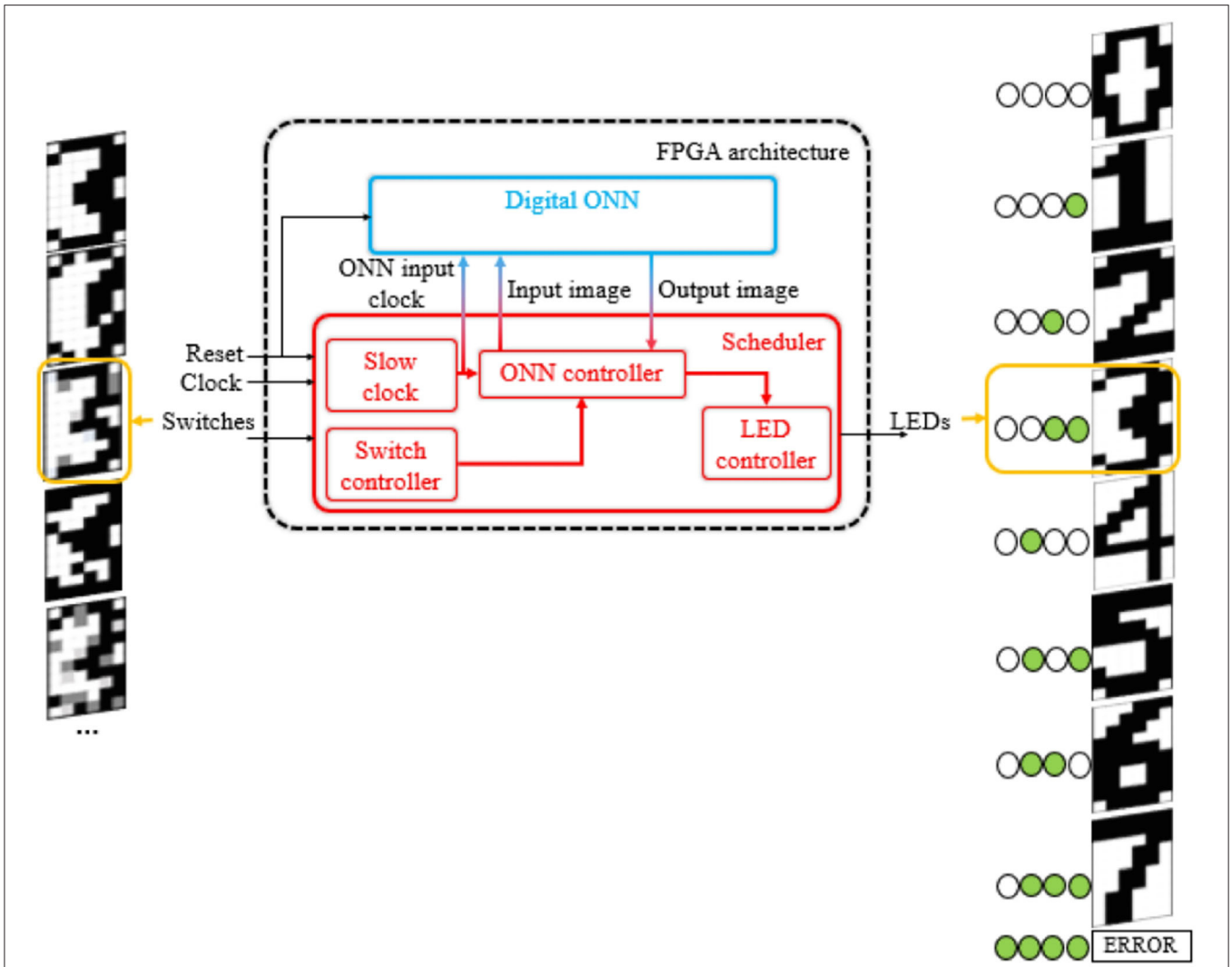

FIGURE 6 | ONN FPGA implementation architecture for pattern recognition on Zybo-Z7 development board.

TABLE 1 | Pattern combinations used for implementation characterization.

ONN size	Number of stored patterns	Patterns
5 × 3	2	0, 1
5 × 3	2	0, 2
5 × 3	2	1, 2
5 × 3	3	0, 1, 2
10 × 6	3	0, 1, 2
10 × 6	4	0, 1, 2, 3
10 × 6	5	0, 1, 2, 3, 4
10 × 6	6	0, 1, 2, 3, 4, 5
10 × 6	7	0, 1, 2, 3, 4, 5, 6
10 × 6	8	0, 1, 2, 3, 4, 5, 6, 7

with ϵ, the number of errors and I_{tests}, the number of test images. We consider an output as an error when the retrieved pattern does not correspond to any stored ones or when the ONN does not stabilize (inconsistent signal activated).

Next, we experimentally measure initialization time (t_{init}), and computation time (t_{comp}) of the ONN with an oscilloscope to calculate the number of frames per second (*FPS*) that ONN implemented on FPGA can treat. It is calculated as:

$$FPS = \frac{1}{t_{init} + t_{comp}} \tag{9}$$

where initialization is the time needed to send serially the test pattern to the ONN. Thus, initialization time grows linearly with the increase of the ONN size.

2.5. Digits Recognition Application Methods

To prove the ONN's capability to perform real-world applications, a 10 × 6 ONN is implemented on FPGA

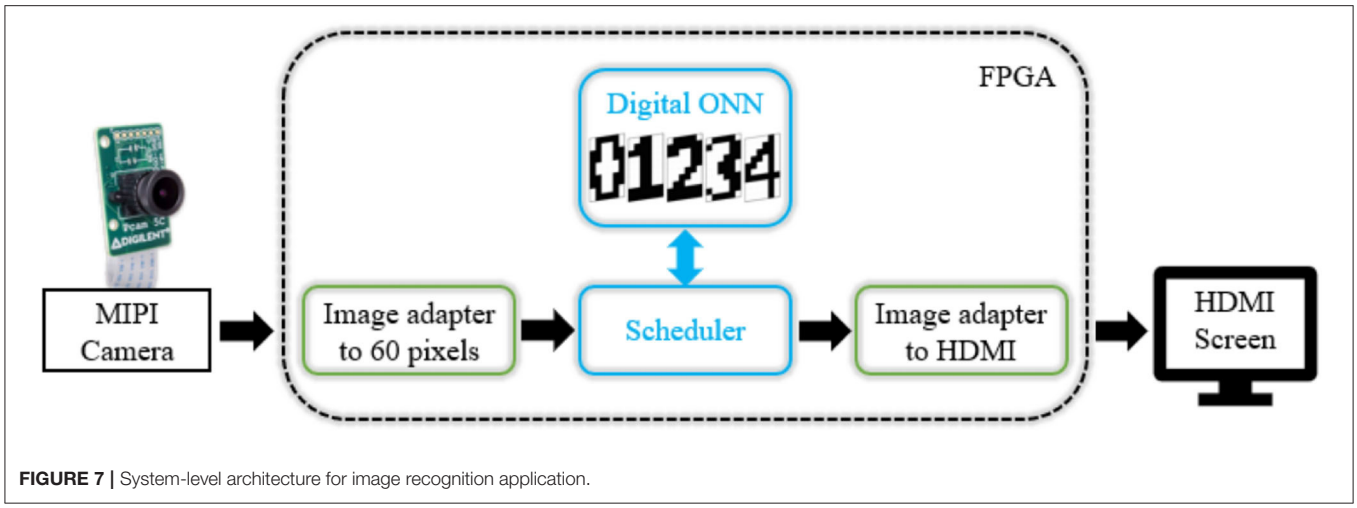

FIGURE 7 | System-level architecture for image recognition application.

inside a complete image recognition system with a camera stream.

We configure a 10 × 6 ONN to recognize digits. We use a camera to stream digits as the input image for ONN. Input images are displayed by a phone to the camera with a dedicated application. The camera is connected to the development board and sends input images to the ONN. When the computation time is over, the output pattern is displayed on an external screen (see **Figure 7**).

We use a Pcam 5c (Digilent, 2017) camera which is connected via a MIPI CSI-2 (MIPI Camera Serial Interface 2) with the Zybo-Z7 development board. We connect the external screen via HDMI (High Definition Multimedia Interface) communication. The image streaming from the camera to the screen comes from a Digilent Github project named Zybo-Z7-Pcam-5c (Digilent, 2020), compatible with the Xilinx's Vivado software 2018.2. We embed the digital ONN inside the image treatment flow. To do so, we convert the camera's image in greyscale, binarize it in black/white pixels, and scale it down to a 10 × 6 pixels image taking the primary color between black/white pixels. We rescale ONN output into a 1280x720 pixels image to display it on the screen. Both rescaling steps use the Vivado HLS tool from Xilinx. We also use development board processor resources to configure the camera. Based on our characterization results, we can parametrize the ONN to a certain training configuration. We choose to train the 10 × 6 ONN to recognize five digits, from 0 to 4, with the Hebbian learning rule. The test set comprises five trained images and 20 corrupted images, similar to ONN characterization, so we expect equal results.

3. RESULTS

3.1. Characterization Results

We validate and characterize digital ONN design with pattern recognition configurations described in section 2.4 and analyze ONN performances. We characterize 5 × 3 and 10 × 6 digital ONNs using both Hebbian and Storkey learning rules. First, in section 3.1.1, we show simulation results and ONN performances. Then, in section 3.1.2, we detail results obtained with the ONN FPGA implementation.

TABLE 2 | Frequency limit and resource utilization estimated in simulation for Xilinx 7-series FPGA.

Design	Maximum frequency	LUTs (%)	Flip-Flops (%)
5 × 3 - Hebbian	83.33 MHz	958 (1.8)	721 (0.68)
5 × 3 - Storkey	83.33 MHz	800 (1.5)	721 (0.68)
10 × 6 - Hebbian	64.10 MHz	6,426 (12.08)	2,756 (2.59)
10 × 6 - Storkey	60.61 MHz	6,192 (11.64)	2,756 (2.59)

3.1.1. Simulation

We simulate 5 × 3 and 10 × 6 ONNs and report on pattern retrieval for Hebbian and Storkey learning rules. Simulation tests of the 5 × 3 ONN using Hebbian and Storkey learning rules result in only 1 test pattern not correctly retrieved. In contrast, simulation tests of the 10 × 6 ONN have different results with the Hebbian learning rule or the Storkey learning rule. Using Hebbian, 5 test images out of 25 do not converge precisely to their respective stored pattern. However, using Storkey, the ONN design retrieves the whole test set successfully. It confirms Storkey's better storage capacity mentioned in section 2.2. We achieve pattern recognition task, with a worst case of 5 images not retrieved (errors) out of 25 (20% error rate).

Additional post place&route simulation allows us to extract estimated characteristics about maximum operating frequency and required resources, see **Table 2**. Such results indicate that both frequency and logical resources are highly dependent on the number of neurons. The smaller the network size is, the higher the system clock frequency can be. Note that frequency differs for the 10 × 6 ONN design that uses Hebbian or Storkey learning rules. Frequency is limited by the synapses block, and synaptic weights are different when using one or another learning rule. It changes the critical path length leading to different frequency limits. In any case, note that differences are almost insignificant.

TABLE 3 | Resource utilization reported for multiple ONN size for Xilinx 7-series FPGA.

#Neurons	#Synapses	LUTs (%)	Flip-Flops (%)
15	225	900 (1.7)	721 (0.68)
60	3,600	6,300 (12)	2,756 (2.59)
100	10,000	30,033 (56)	4,985 (5)
120	14,400	38,372 (72)	5,970 (6)
140	19,600	46,900 (88)	6,955 (7)
150	22,500	65,251 (123)	7,447 (7)

Besides, the 5 × 3 ONN design requires nearly ten times fewer resources than the 10 × 6 ONN design. It highlights one of the digital ONN design limits. An increase in the ONN size extends ONN logical resources. Depending on the used FPGA, the number of neurons will be limited. In the next experiments, we use a Xilinx-7 series FPGA. **Table 3** details resource utilization for multiple ONN sizes on the Xilinx 7-series FPGA. It shows the increase in LUTs with ONN size. For the given ONN design, we can implement stand-alone ONNs between 140 and 150 neurons.

These first simulation results confirm the digital ONN capability to perform pattern recognition.

3.1.2. FPGA Implementation

We validate and characterize our digital ONN implementation by reporting on ONN pattern recognition accuracy (error rate) for multiple parameters. First, we use a low frequency and we perform the same tests as in simulation to check the ONN FPGA implementation. Then, we compare the error rate for multiple training configurations. Next, we observe the error rate for faster frequencies to check the frequency limit. Finally, we measure the computation time and calculate the number of FPS treatable by the ONN.

3.1.2.1. ONN Training Configuration

Here, we present the ONN operation for multiple training configurations. Results are shown in **Table 4** for the 5 × 3 ONN and the 10 × 6 ONN. First, we validate the ONN implementation by comparing results with simulation tests. Implementation tests with the same training configuration as simulation tests give equal results. Then, we observe that the error rate increases with the number of stored patterns for both learning rules. With the 5 × 3 ONN, we obtain similar results with both Hebbian and Storkey learning rules. However, with the 10 × 6 ONN, we obtain significant error rate differences. We notice that weights trained with Storkey give a lower error rate than weights trained with Hebbian for four pattern combinations out of five. Observations also indicate which training configuration can be the best option for a particular application considering an acceptable error rate. If we consider 0% acceptable error rate for the 10 × 6 ONN, we can store patterns from 0 to 3 with the Hebbian learning rule, and we can add digit 4 if we use the Storkey learning rule.

3.1.2.2. ONN Frequency

Simulation reveals ONN maximum frequency depends on the applied learning rule. With FPGA implementation, ONN input frequency choice is limited. We perform FPGA implementation experiments on the same range of frequencies to check if implementation and simulation results match. Experiments on 5 × 3 ONN perform similarly at 7.8125, 62.5, and 125 MHz as shown in **Figure 8A**. Thus, ONN can run a given test set at higher frequencies than the evaluated limit (83, 33 MHz) for all tested training configurations. The difference in frequencies can be explained by the way they are measured—in simulation, ONN frequency was evaluated with global static timing analysis, whereas in experiments, frequency is evaluated on a specific test set. **Figure 8B** shows the 10 × 6 ONN error rate at different frequencies for four training configurations. We observe a trade-off between error rate, operating frequency, and training configuration. Also, we note that for each training configuration, only a frequency of 125 MHz impacts the 10 × 6 ONN error rate.

Considering simulation and implementation results, we assess the 5 × 3 ONN on FPGA maximum operating frequency to be 62.5 MHz, and the 10 × 6 ONN on FPGA maximum operating frequency to be 31.25 MHz. In our next experiments, related to time measurements, we define a common digital ONN input frequency for 5 × 3 and 10 × 6 ONNs. We set it to 31.25 MHz as it is the highest common operating frequency.

3.1.2.3. ONN Computation Time

We assess the computation speed of our ONN implementation with time measurements. We experimentally measure the ONN initialization time needed to apply the input image to ONN, and the computation time, from the end of the initialization process to the *steady* signal's activation time (see section 2.3). We measure ONN timings for multiple training configurations shown in **Table 5**. We choose training configurations that showed a 0% error rate to avoid the maximum computation time clamped to 160 μs when ONN does not converge. We measure the computation time as the time required for the ONN to reach a stable correct output pattern (0% error rate). We set the digital ONN input frequency to 31.25 MHz.

Computation time, initialization time, and FPS results are listed in **Table 5**. We initialize the ONN by presenting data serially to the ONN, so it depends on the size of the ONN, regardless of the applied learning rule. It explains why the initialization time is longer for 10 × 6 than 5 × 3 ONN.

We observe that ONN computation time does not vary and slightly increases with ONN size. It is an attractive feature of the ONN concept in which convergence is achieved in a few oscillation cycles independently of the number of neurons. It is also worth mentioning that the degradation of FPS performance is due to the initialization time that grows linearly with the number of neurons because of its serial implementation. It could be mitigated by using a different initialization approach.

Simulation and implementation characterizations allowed us to validate our ONN digital design to perform pattern recognition with a minimum error rate of 0%. We performed multiple experiments to highlight the advantages and limitations of the digital ONN. The main limit concerns the digital ONN FPGA resources (LUTs, Flip-Flops). The current digital ONN design implemented on the $XC7Z020 - 1CLG400C$ FPGA is

TABLE 4 | Experimental error rate of the 5 × 3 and the 10 × 6 ONN implemented on FPGA for various training configurations.

ONN	Stored patterns	Learning rule	Test images	Errors	Error rate (%)
5 × 3	0,1	Hebbian	10	0	0
		Storkey	10	0	0
5 × 3	0,2	Hebbian	10	0	0
		Storkey	10	0	0
5 × 3	1,2	Hebbian	10	0	0
		Storkey	10	0	0
5 × 3	0,1,2	Hebbian	15	1	6.67
		Storkey	15	1	6.67
10 × 6	0,1,2,3	Hebbian	20	0	0
		Storkey	20	0	0
10 × 6	0,1,2,3,4	Hebbian	25	5	20
		Storkey	25	0	0
10 × 6	0,1,2,3,4,5	Hebbian	30	9	30
		Storkey	30	1	3
10 × 6	0,1,2,3,4,5,6	Hebbian	35	21	60
		Storkey	35	4	11
10 × 6	0,1,2,3,4,5,6,7	Hebbian	40	35	87.5
		Storkey	40	4	10

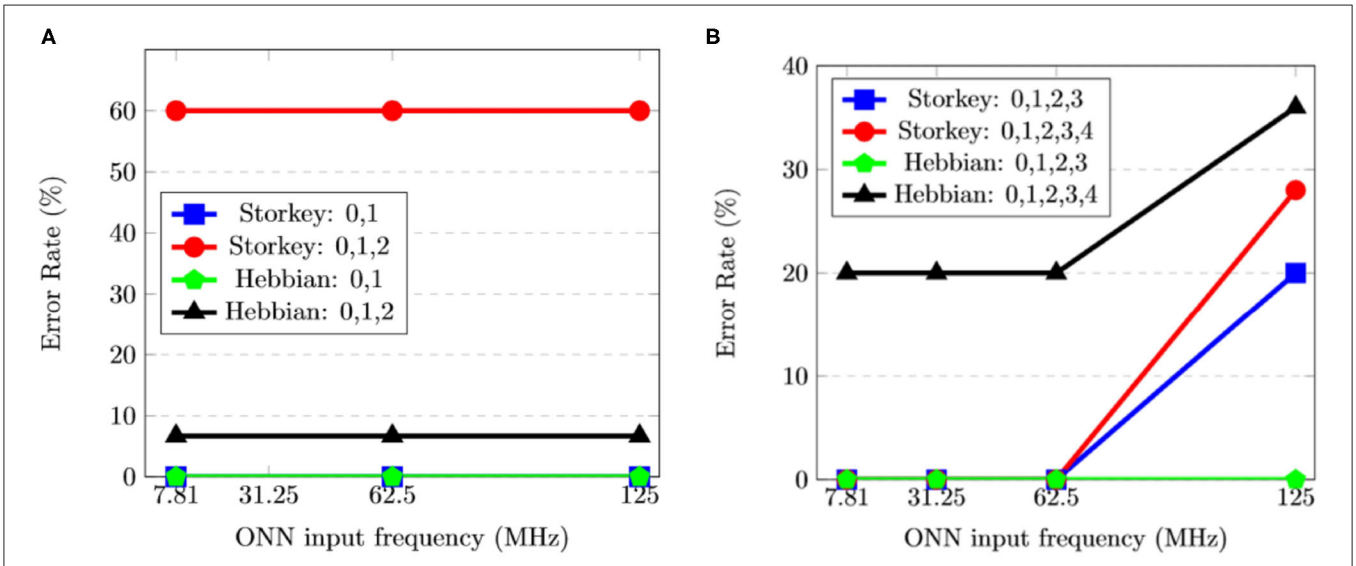

FIGURE 8 | Experimental results of the error rate (%) for various frequencies (MHz) and training configurations (learning rule and pattern combination) for **(A)** the 5 × 3 ONN, and **(B)** the 10 × 6 ONN.

limited to a 100 neurons. In contrast, we highlighted the short-time computation required by the digital ONN. We can compute each pattern with an average of 5 μs at 31.25 MHz frequency for both 5 × 3 and 10 × 6 ONNs. So, ONN size does not impact the computation time, which is an important feature of ONN to further explore design methods to upscale its size.

TABLE 5 | ONN timing performances for various sizes (number of neurons) and learning rules.

Learning rule	ONN	Stored patterns	Initialization time (us)	Computation time, avg (us)	FPS
Hebbian	5 × 3	[0,1]	2	5.04	142,045
Hebbian	10 × 6	[0,1,2,3]	7.8	5.2	76,923
Storkey	5 × 3	[0,1]	2	5.05	141,844
Storkey	10 × 6	[0,1,2,3]	7.8	5.4	75,757

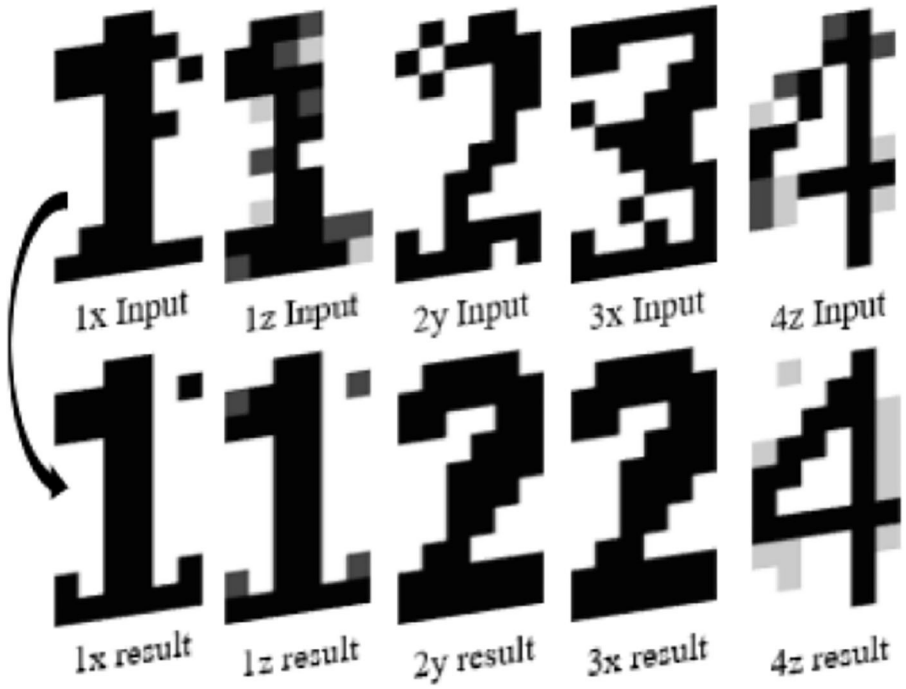

FIGURE 9 | Representation of error images for 10 × 6 ONN digit recognition application showing incorrect pixels.

3.2. Digits Recognition Application Results

We validate the image recognition application by comparing errors with previous characterization tests. We use the output HDMI screen to identify recognized images and errors of the digital ONN image recognition application. We expect results to match with characterization ones and as expected, we find five images not correctly recognized, see **Figure 9**. Output patterns displayed on the screen reveal that for each not-correctly recognized image, ONN is close to a correct reference pattern with only a few pixels wrong. However, the reference pattern is not necessarily the expected one, as for the image 3x. We expected a 3, but the result is closer to a 2. Some errors can be explained by test images that correspond to corrupted digits too far in their HD from reference patterns. With this application, we demonstrate the feasibility of using an ONN inside a complete design. Also, we reach 20% error rate with this application using Hebbian weights, but we know thanks to characterization, that Storkey weights can reach 0% error rate. Besides, we know that 10 × 6 ONN takes 7.8 μs to be initialized, 5 μs to stabilize on average, and 160 μs if it does not stabilize. As the camera provides an image every 15 *ms*, the 10 × 6 ONN does not create any latency for the image recognition application, so it respects real-time requirements. It validates our digital ONN design as a solution for image recognition applications and encourages us to look into other embedded applications.

4. DISCUSSION

Our work presents the design and performances of a novel fully digital ONN implementation. Further, we demonstrate ONN on FPGA implementation for image recognition applications. It is a proof-of-concept on the potential of ONN as an alternative computing paradigm. Here, we present the digital ONN design aspects and their advantages, limits, and future directions.

TABLE 6 | Frequency, computation time and resource comparison of ONN and HNN designs.

Neural network	Size	Frequency (MHz)	Computation time (us) Max - Avg	Computation time (clock cycles) Max - Avg	Resources (LUTs)
HNN (*)	16	81.39	x - 1.3	x - 100	390
HNN (*)	32	33.55	x - x	x - x	700
ONN					
- Hebbian	15	83.33	2.74 - 0.92	228 - 77	958
- Storkey	15	83.33	1.18 - 0.77	98 - 64	800
ONN					
- Hebbian	60	64.1	3.53 - 1.44	226 - 92	6,426
- Storkey	60	60.61	1.75 - 1.18	106 - 72	6,192

(*) Refers to De Abreu de Sousa et al. (2014). Note, HNN resource estimation comes from data extraced from De Abreu de Sousa et al. (2014) and FPGA documentation. Also note, HNNs are implemented on a Xilinx 3-series FPGA (4-input LUTs) while ONNs are implemented on a Xilinx 7-series FPGA (6-input LUTs).

4.1. Advantages

Presently, we developed ONN designs that exhibit good performances with 5 × 3 and 10 × 6 ONNs.

An interesting feature of our approach in comparison with Jackson et al. (2019) is that by resorting to the 1-bit oscillation at the neuron's output, multipliers are avoided in the synapses block, while still retaining a multi-level neuron. This is possible because we encode the state in the neuron's oscillation phase.

Another advantage of our approach is the easiness of training. Results reported in this paper have been obtained using simple Hebbian and Storkey rules. Given the patterns to be stored/recognized, weights are calculated offline by using matrix operations, while training other neural models can be a very time-consuming operation. Limited retrieval capacity has been obtained in some of the experiments described. However, the accuracy of the ONN can be increased by resorting to enhanced learning rules.

Finally, a fundamental advantage of ONN is the fast computation. ONN parallel behavior allows the independence of the computation time and the network size. In this paper, we achieved more than 70000 FPS with 10 × 6 ONN and a serial initialization of the neurons.

4.2. Limitations and Future Directions

The present digital ONN design has also limitations.

First, our ONN is a preliminary design developed to validate the concept, and different optimization procedures at different levels are currently being carried out.

Second, the limited FPGA resources introduce some constraints on the ONN size that can be implemented on FPGA (see **Table 3**). Such limited resources are the computation resources (LUTs). They are used to implement the combinatorial synapses block whose size increases quadratically with the number of neurons.

The limited size does not allow for comparison with standard benchmark sets usually used to evaluate neural networks. For example, comparing ONN with SNN is not trivial because of the paradigm differences, such as the different network architectures and learning algorithms. To make a meaningful comparison, a common ground is necessary with a common application and benchmark. Comparison to other reported similar FPGA-based implementations of neural networks is meaningful only if the same ONN size are developed.

The Associative Memory Neural Networks (AMNNs), such as HNNs, are the closest ANNs that can be compared with ONNs. In the existing literature, we found digital implementations of AMNNs such as Leiner et al. (2008), Mansour et al. (2011), and De Abreu de Sousa et al. (2014). The most relevant comparisons can be made with the digital design from De Abreu de Sousa et al. (2014)'s work, which is also the most recent one. In this work, authors perform a frequency study for multiple HNN sizes, from 16 neurons to 32 neurons. Stored patterns resemble our stored patterns representing digits. For example, they use 8 × 4 representations of the letter U and the number 5. Tests are also similar as they use stored and corrupted patterns as inputs.

Table 6 shows the comparison between (De Abreu de Sousa et al., 2014)'s HNN and our ONN. Results show that ONN has a higher operating frequency than HNN. Such as in the 32 neurons case, the maximum HNN frequency is 33.55 MHz, but ONN can run faster, up to 60.61 MHz for 10 × 6 ONN. We do not have information about the computation time for the 32-neuron HNN, however, if we compare the 16-neuron HNN with the 5 × 3 ONN, we expect similar trends in frequencies and computation times. Though it is difficult to make any conclusive comparisons, it seems that our larger ONN can operate at higher frequencies than the cited HNN.

Motivated by the potential of the proposed ONN, we are currently exploring optimizations in several directions like hardware resources, frequency, and accuracy. For example, using a faster internal clock than the oscillator frequency, a sequential implementation could be used to reduce the size of this resource-consuming block. Also, we can explore additional learning rules to increase accuracy. At this time, we have only studied Hebbian and Storkey, which are local and incremental with a limited storage capacity, but other learning rules will also be explored to have a better assessment of learning rules suitable

for ONNs. For example, the pseudo-inverse rule (also called projection rule) can increase HNNs storage capacity and improve accuracy (Wu et al., 2012; Sahoo et al., 2016), but is not local nor incremental. Moreover, recent works have explored learning rules with self-feedback connections (non-0 diagonal), and have shown higher accuracy for a high number of stored patterns (Liou and Yuan, 1999; Folli et al., 2017; Rocchi et al., 2017; Gosti et al., 2019). In summary, despite the present limitations of the ONN, features in terms of FPS, computation time and training, are encouraging toward the exploration of a wider range of applications.

5. CONCLUSION

In this paper, we carried out the questions—can we use ONN for image recognition, and—what are the advantages and limitations of ONN for AI at-the-edge applications. To do so, we presented a proof of concept of the ONN neuromorphic computing paradigm with a fully digital design. We validated the computing capability of a 5 × 3 ONN and a 10 × 6 ONN performing pattern recognition both in simulation and FPGA implementation. We used Hebbian or Storkey learning rules to train our ONN. For both learning rules, with three stored patterns, the 5 × 3 ONN retrieved 14 test patterns out of a test set of 15. For the 10 × 6 ONN with five stored patterns, results differ from Hebbian and Storkey learning rules. ONN with weights computed with Storkey can retrieve 25 test patterns out of 25, but using Hebbian ONN can only retrieve 20 test patterns out of 25. Further experiments confirmed that Storkey is more accurate than Hebbian for an equal number of stored patterns. We performed additional experiments to characterize our digital ONN. First, we estimated by simulation a maximum operating frequency for the 5 × 3 ONN to 83,33 MHz. Then, we showed that for a specific application, the ONN implemented on FPGA could go up to 125 MHz, without any changes on the ONN operation. Besides, we performed a timing analysis on digital ONNs. We measured the initialization time needed to apply the input pattern to the ONN, and the computation time, needed by the ONN to stabilize to a stored pattern. From measurements, we were able to calculate the maximum FPS (Frames per second). For the 10 × 6 ONN, we obtained a maximum FPS around 76000, at 31.25 MHz with a training configuration resulting in all test patterns successfully retrieved. Then, we embedded the 10 × 6 ONN into a complete image recognition application performing digit recognition from a camera stream. It respected real-time constraints, and we demonstrated that the ONN paradigm can fit with AI at-the-edge image recognition application. Thus, despite the size limitation (about a 100 neurons) of our digital design due to high FPGA resource consumption, the huge potentiality of ONN is undeniable. ONNs are still in their infancy for comparison with standard benchmarks and it is the focus of future works. The potential of the proposed ONN propels further investigation to explore its capabilities on diverse applications for AI at-the-edge.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

AUTHOR CONTRIBUTIONS

AT-S motivated the project and experiments. MJ, JN, and MJA conducted the digital ONN design development and simulation characterization. MA and TGi developed the ONN design implementation on FPGA, image recognition application, and performed all measurements. BL-B, TGo, and TH were involved in the discussion and editing of the manuscript and provided valuable inputs at multiple stages of this work. All authors contributed to the article and approved the submitted version.

FUNDING

This work is supported by the European Union's Horizon 2020 research and innovation program, EU H2020 NEURONN (www.neuronn.eu) project under grant no. 871501.

REFERENCES

Ahmed, I., Chiu, P.-W., Moy, W., and Kim, C. H. (2021). A probabilistic compute fabric based on coupled ring oscillators for solving combinatorial optimization problems. *IEEE J. Solid State Circ*. doi: 10.1109/JSSC.2021.3062821

Amit, D. J., Gutfreund, H., and Sompolinsky, H. (1987). Statistical mechanics of neural networks near saturation. *Ann. Phys*. 173, 30–67. doi: 10.1016/0003-4916(87)90092-3

Bey (2020). Beyond von Neumann. *Nat. Nanotechnol*. 15:507. doi: 10.1038/s41565-020-0738-x

Corti, E., Gotsmann, B., Moselund, K., Stolichnov, I., Ionescu, A., and Karg, S. (2019). "Resistive coupled VO2 oscillators for image recognition," in *2018 IEEE International Conference on Rebooting Computing, ICRC 2018* (Tysons, VA: Institute of Electrical and Electronics Engineers Inc.).

Csaba, G., and Porod, W. (2013). Computational study of spin-torque oscillator interactions for non-Boolean computing applications. *IEEE Trans. Magn*. 49, 4447–4451. doi: 10.1109/TMAG.2013.2244202

Csaba, G., and Porod, W. (2020). Coupled oscillators for computing: a review and perspective. *Appl. Phys. Rev*. 7:011302. doi: 10.1063/1.5120412

Csaba, G., Ytterdal, T., and Porod, W. (2016). "Oscillatory neural network from ring oscillators," in *CNNA 2016; 15th International Workshop on Cellular Nanoscale Networks and their Applications* (Dresden), 1–2.

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

De Abreu de Sousa, M. A., Horta, E. L., Kofuji, S. T., and Del-Moral-Hernandez, E. (2014). Architecture analysis of an FPGA-based hopfield neural network. *Adv. Artif. Neural Syst*. 2014, 1–10. doi: 10.1155/2014/602325

Digilent (2017). *Pcam 5C Reference Manual*. Available online at: https://reference.digilentinc.com/add-ons/pcam-5c/reference-manual

Digilent (2018). *Zybo Z7 Reference Guide*. Available online at: https://reference.digilentinc.com/_media/reference/programmable-logic/zybo-z7/zybo-z7_rm.pdf

Digilent (2020). *Zybo Z7 -20 Pcam 5C Demo*. Available online at: https://github.com/Digilent/Zybo-Z7-20-pcam-5c doi: 10.36311/1519-0110.2019.v20n1.01.p5

Folli, V., Leonetti, M., and Ruocco, G. (2017). On the maximum storage capacity of the hopfield model. *Front. Comput. Neurosci*. 10:144. doi: 10.3389/fncom.2016.00144

Gosti, G., Folli, V., Leonetti, M., and Ruocco, G. (2019). Beyond the maximum storage capacity limit in hopfield recurrent neural networks. *Entropy* 21:726. doi: 10.3390/e21080726

Guo, W., Yantir, H. E., Fouda, M. E., Eltawil, A. M., and Salama, K. N. (2021). Toward the optimal design and fpga implementation of spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst*. 11, 1–15. doi: 10.1109/TNNLS.2021.3055421

Han, J., Li, Z., Zheng, W., and Zhang, Y. (2020). Hardware implementation of spiking neural networks on fpga. *Tsinghua Sci. Technol*. 25, 479–486. doi: 10.26599/TST.2019.9010019

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. U.S.A*. 79, 2554–2558. doi: 10.1073/pnas.79.8.2554

Hoppensteadt, F. C., and Izhikevich, E. M. (1997). *Neural Networks*. New York, NY: Springer New York.

Hoppensteadt, F. C., and Izhikevich, E. M. (2000). Pattern recognition via synchronization in phase-locked loop neural networks. *IEEE Trans. Neural Netw*. 11, 734–738. doi: 10.1109/72.846744

Jackel, L. D., Stenard, C. E., Baird, H. S., Boser, B., Bromley, J., Burges, C. J., et al. (1991). "A neural network approach to handprint character recognition," in *Digest of Papers-IEEE Computer Society International Conference* (San Francisco, CA: IEEE), 472–475.

Jackson, T., Pagliarini, S., and Pileggi, L. (2019). "An oscillatory neural network with programmable resistive synapses in 28 Nm CMOS," in *2018 IEEE International Conference on Rebooting Computing, ICRC 2018* (Tysons, VA: Institute of Electrical and Electronics Engineers Inc.).

Jackson, T. C., Sharma, A. A., Bain, J. A., Weldon, J. A., and Pileggi, L. (2015). "An RRAM-based oscillatory neural network," in *2015 IEEE 6th Latin American Symposium on Circuits and Systems, LASCAS 2015-Conference Proceedings*. (Montevideo: Institute of Electrical and Electronics Engineers Inc.).

Jiao, L., Zhang, F., Liu, F., Yang, S., Li, L., Feng, Z., et al. (2019). A survey of deep learning-based object detection. *IEEE Access* 7:128837–128868. doi: 10.1109/ACCESS.2019.2939201

Kendall, J. D., and Kumar, S. (2020). The building blocks of a brain-inspired *Computer* 7, 011305. doi: 10.1063/1.5129306

Khan, M., Lester, D., Plana, L., Rast, A., Jin, X., Painkras, E., et al. (2008). "Spinnaker: mapping neural networks onto a massively-parallel chip multiprocessor," in *2008 IEEE International Joint Conference on Neural Networks* (Hong Kong: IEEE World Congress on Computational Intelligence), 2849–2856.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Commun ACM* 60, 84–90. doi: 10.1145/3065386

Kumar, A., and Mohanty, P. (2017). Autoassociative memory and pattern recognition in micromechanical oscillator network. *Sci. Rep*. 7:411. doi: 10.1038/s41598-017-00442-y

Leiner, B. J., Lorena, V. Q., Cesar, T. M., and Lorenzo, M. V. (2008). "Hardware architecture for FPGA implemetation of neural network and its application in images processing," in *Proceedings-2008 4th Southern Conference on Programmable Logic, SPL* (Bariloche: IEEE Computer Society), 209–212.

Levi, T., Nanami, T., Tange, A., Aihara, K., and Kohno, T. (2018). Development and applications of biomimetic neuronal networks toward brainmorphic artificial intelligence. *IEEE Trans. Circ. Syst. II Express Briefs* 65, 577–581. doi: 10.1109/TCSII.2018.2824827

Lieff, J. (2012). *What is Mind? Brain Oscillations, Synchronous Brain Waves and Consciousness*. Available online at: https://jonlieffmd.com/blog/what-is-mind-consciousnesshuman-brain-oscillations-and-synchronous-brain-waves

Liou, C.-Y., and Yuan, S.-K. (1999). Error tolerant associative memory. *Biol. Cybern*. 81:331–342. doi: 10.1007/s004220050566

Ma, D., Shen, J., Gu, Z., Zhang, M., Zhu, X., Xu, X., et al. (2017). Darwin: a neuromorphic hardware co-processor based on spiking neural networks. *J. Syst. Arch*. 77, 43–51. doi: 10.1016/j.sysarc.2017.01.003

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Netw*. 10, 1659–1671. doi: 10.1016/S0893-6080(97)00011-7

Mansour, W., Ayoubi, R., Ziade, H., Velazco, R., and EL Falou, W. (2011). An optimal implementation on FPGA of a hopfield neural network. *Adv. Artif. Neural Syst*. 2011:1–9. doi: 10.1155/2011/189368

Martindale, C., and H. N. (1978). Eeg differences as a function of creativity, stage of the creative process, and effort to be original. *Biol. Psychol*. 6, 157–167. doi: 10.1016/0301-0511(78)90018-2

Met (2013). Spontaneous synchronization. UCLA Department of physics and astronomy. Available online at: https://www.youtube.com/watch?v=T58lGKREubo

Misra, J., and Saha, I. (2010). Artificial neural networks in hardware: a survey of two decades of progress. *Neurocomputing* 74, 239–255. doi: 10.1016/j.neucom.2010.03.021

Mitra, S., and Fusi, S., I. G. (2009). Real-time classification of complex patterns using spike-based learning in neuromorphic vlsi. *IEEE Trans. Biomed. Circ. Syst*. doi: 10.1109/TBCAS.2008.2005781

Morris, R. G. (1999). D.O. Hebb: the organization of behavior, Wiley: New York; 1949. *Brain Res. Bull*. 50:437. doi: 10.1016/S0361-9230(99)00182-3

Muezzinoglu, M., Guzelis, C., and Zurada, J. (2003). A new design method for the complex-valued multistate Hopfield associative memory. *IEEE Trans. Neural Netw*. 14, 891–899. doi: 10.1109/TNN.2003.813844

Nassif, A. B., Shahin, I., Attili, I., Azzeh, M., and Shaalan, K. (2019). Speech recognition using deep neural networks: a systematic review. *IEEE Access* 7, 19143–19165. doi: 10.1109/ACCESS.2019.2896880

Paugam-Moisy, H., and Bohte, S. (2012). *Computing With Spiking Neuron Networks*. Berlin; Heidelberg: Springer Berlin Heidelberg.

Pham, H., Nguyen, M., and Sun, C. (2019). "Aiot solution survey and comparison in machine learning on low-cost microcontroller," in *2019 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)* (Taipei), 1–2.

Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Adler, T., et al. (2021). Hopfield networks is all you need. *arXiv:2008.02217*. Available online at: https://arxiv.org/abs/2008.02217

Raychowdhury, A., Parihar, A., Smith, G. H., Narayanan, V., Csaba, G., Jerry, M., et al. (2019). Computing with networks of oscillatory dynamical systems. *Proc. IEEE* 107, 73–89. doi: 10.1109/JPROC.2018.2878854

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). "You only look once: unified, real-time object detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Vol. 2016 (Las Vegas, NV: IEEE Computer Society), 779–788.

Rocchi, J., Saad, D., and Tantari, D. (2017). High storage capacity in the hopfield model with auto-interactions—stability analysis. *J. Phys. A Math. Theor*. 50, 465001. doi: 10.1088/1751-8121/aa8fd7

Rosado-Muñoz, A., Bataller-Mompeán, M., and Guerrero-Martínez, J. (2012). FPGA implementation of spiking neural networks. *IFAC Proc*. 45, 139–144. doi: 10.3182/20120403-3-DE-3010.00074

Roychowdhury, J. (2014). Boolean computation using self-sustaining nonlinear oscillators. *CoRR, abs/1410.5016*.

Sahoo, R. C., Kumar, S., and Goswami, P. (2016). Implementation of hopfield neural network for its capacity with finger print images. *Int. J. Comput. Appl*. 141, 44–49. doi: 10.5120/ijca2016909625

Shah, M., and Kapdi, R. (2017). "Object detection using deep neural networks," in *Proceedings of the 2017 International Conference on Intelligent Computing and Control Systems, ICICCS 2017*, Vol. 2018 (Madurai: Institute of Electrical and Electronics Engineers Inc.). 787–790.

Shi, R., Jackson, T. C., Swenson, B., Kar, S., and Pileggi, L. (2016). "On the design of phase locked loop oscillatory neural networks: mitigation of transmission delay effects," in *Proceedings of the International Joint Conference on Neural Networks*, Vol. 2016 (Vancouver, BC: Institute of Electrical and Electronics Engineers Inc.), 2039–2046.

Shukla, N., Tsai, W.-Y., Jerry, M., Barth, M., Narayanan, V., and Datta, S. (2016). "Ultra low power coupled oscillator arrays for computer vision applications," in *2016 IEEE Symposium on VLSI Technology* (Honolulu, HI), 1–2.

Storkey, A. (1997). "Increasing the capacity of a hopfield network without sacrificing functionality," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 1327, (Laussane: Springer Verlag), 451–456.

Storkey, A., Storkey, A., and Valabregue, R. (1997). A Hopfield learning rule with high capacity storage of time-correlated patterns. *Electron. Lett*. 33, 1803–1804. doi: 10.1049/el:19971233

Tanaka, G., and Aihara, K. (2009). Complex-valued multistate associative memory with nonlinear multilevel functions for gray-level image reconstruction. *IEEE Trans. Neural Netw.* 20, 1463–1473. doi: 10.1109/TNN.2009.2025500

Velichko, A., Belyaev, M., and Boriskov, P. (2019). A model of an oscillatory neural network with multilevel neurons for pattern recognition and computing. *Electronics* 8, 75. doi: 10.3390/electronics8010075

Wu, Y., Hu, J., Wu, W., Zhou, Y., and Du, K. L. (2012). "Storage capacity of the hopfield network associative memory," in *Proceedings-2012 5th International Conference on Intelligent Computation Technology and Automation, ICICTA* 2012, (Zhangjiajie) 330–336.

Xia, Y., Levi, T., and Kohno, T. (2020). Digital hardware spiking neuronal network with stdp for real-time pattern recognition. *J. Rob. Netw. Artif. Life* 7, 121–124. doi: 10.2991/jrnal.k.200528.010

Xiong, W., Wu, L., Alleva, F., Droppo, J., Huang, X., and Stolcke, A. (2018). "The microsoft 2017 conversational speech recognition system," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing-Proceedings*, Vol. 2018 (Institute of Electrical and Electronics Engineers Inc.), 5934–5938.

Yang, T., and Song, J. (2018). "An automatic brain tumor image segmentation method based on the u-net," in *2018 IEEE 4th International Conference on Computer and Communications, ICCC 2018* (Salamanca: Institute of Electrical and Electronics Engineers Inc.), 1600–1604.

Zenke, F., and Ganguli, S. (2018). Superspike: supervised learning in multilayer spiking neural networks. *Neural Comput.* 30, 1514–1541. doi: 10.1162/neco_a_01086

Conflict of Interest: TGo and TH were employed by the company A.I. Mergence, Paris.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.